03

Tratando el error 404 #1

Transcripción

[00:00] Bien, entonces ya vimos aquí que el problema que tenemos ahora es este código de estado. ¿Y por qué? Porque 500 no es el mejor código para retornar en este caso. Si yo te dijera que está en el rango de errores del servidor, esto no es un error de mi servidor en realidad porque el recurso no existe.

[00:21] Debería ser un error de que el recurso no fue encontrado. Y si yo te digo que es un error relativo a que un recurso no fue encontrado, ¿cuál error de HTTPS te viene a la mente? Exacto el 404. El error 404 not found sería el error más ideal para este tipo de casos. ¿Cómo hacemos para que Spring me regrese un 404 y no un 500?

[00:46] Vamos al código. Primero vamos a ver lo que está pasando aquí. Aquí en mi terminal puedo ver que la excepción que está haciendo lanzada por Spring data es EntityNotFoundException. ¿Por qué? Porque Spring data lo que dice es select from, como lo vemos aquí, cuando el id sea tal.

[01:08] Y si es que no lo encuentra, lanza una excepción. Esta excepción es atrapada por Spring en un nivel un poco más alto y por lo tanto nos retorna 500, porque para Spring es un error de servidor, y en parte es cierto porque es una excepción que está siendo lanzada, pero nosotros podemos controlar esa excepción porque sabemos qué excepción está siendo lanzada por Spring.

[01:35] Por lo tanto cómo puedo basado en una excepción lanzada por alguna parte de mi API tratar el error y devolver lo que yo quiero. Control tryCatch e

una buena opción, pero no es la más ideal en este caso. ¿Por qué? Miren aquí. El método que está lanzando esta excepción es este de aquí: medicoRepository.getReferenceById.

[02:03] Si yo uso try catch puedo usarlo para tratar el error en este nivel de mi método pero tendría que copiar el mismo try catch en el eliminar médico y en el actualizar médico. Esto ya no es muy inteligente que digamos. ¿Por qué? Porque simplemente no es lo más ideal para hacer, repetir ese código una, una y otra vez.

[02:28] Spring nos da una forma un poco más sencilla y dinámica digamos de poder ejecutar un tratamiento de errores. Para eso necesitamos crear una clase que nos permita mapear los errores que queremos tratar. Regresamos a la estructura de mi proyecto que está aquí, por ejemplo tengo de aquí med.voll controller dirección médico y aquí hay una cosa que yo quisiera que vean aquí, dirección y médico.

[02:55] Y si es que hicieron las actividades del curso anterior, tienen paciente aquí también. Ya no están, digamos, relacionados directamente con el API sino son más entidades de mi dominio, no pertenecen directamente al API sino pertenecen al dominio de mi API.

[03:11] Entonces lo que yo voy a hacer aquí es darle un refactor y las voy a mover a un paquete que se va a llamar domain, dominio. Como fue aquí, hago refactor, y ahora tengo mi dominio con mis dos entidades aquí, están apareciendo algunos errores aquí en mi controller. ¿Esto por qué es?

[03:40] Porque este import que estaba aquí de médico ya no existe, entonces lo único que tengo que hacer es borrarlo y ya el error ya debería desaparecer, vamos a ver dónde más está dando el error. Aquí en datosRegistroMédico, porque déjame ver aquí, exacto. Voy a “Ctrl + Espacio” y ya lo importó correctamente.

[04:04] Entonces ya mis errores están resueltos, ya mi código está compilando nuevamente. Lo único que he hecho como les comenté es crear mi paquete dominio y mover mis dos entidades médico y dirección dentro del paquete dominio. ¿Qué más voy a hacer como parte de mi refactor y para crear la nueva clase que necesito?

[04:30] Voy a darle clic derecho, new package y lo voy a llamar infra, dentro de med.voll.api. Tengo infra, y aquí en infra lo que voy a hacer es crear una nueva clase y esta clase se va a llamar tratadorDeErrores. Lo creo, no lo agrego todavía a Git y listo, por ahora eso es lo que necesito. ¿Por qué?

[04:58] Porque yo quiero tratar mis errores globalmente a nivel de mi controller, de mi proyecto, no a nivel de cada método en específico.